Fondamenti di Informatica T-1, 2019/2020 – Modulo 2

Prova d'Esame 4A di Giovedì 11 Giugno 2020 – tempo a disposizione 2h

Avvertenze per la consegna: apporre all'inizio di ogni file sorgente un commento contenente i propri dati (**cognome, nome, numero di matricola**) e il **numero** della prova d'esame. Al termine, **consegnare tutti i file sorgenti** necessari alla compilazione e alla corretta esecuzione del programma.
Nota: **il main non è opzionale**; i *test* richiesti vanno implementati.
Consiglio: per verificare l'assenza di *warning*, eseguire sempre *"Rebuild All"*.

Valerio gestisce con orgoglio il suo bar Rhythm&Blues ma ultimamente, forse a causa di una moda, sono sempre più i clienti che lo interrogano sulle sue paste e croissant, sull'apporto calorico, e anche sul prezzo. Quindi ha deciso di informatizzare la gestione delle paste a banco. In particolare memorizza in un file di testo le paste disponibili, una pasta su ogni riga, col seguente formato: innanzitutto il **costo** della pasta, ad esempio 1.60 Eur (un float); a seguire, separato da uno spazio, il **tipo** di pasta, ad esempio "crema", "ricotta", "uvetta e crema" (una stringa di al più 255 caratteri utili, contenente spazi); infine, separato da un carattere ';', l'apporto **calorico**, ad esempio 115 Kcal (un intero).
Il nostro barista è intraprendente e ha cominciato a fornire un innovativo servizio di asporto: i clienti gli comunicano il nome di un ingrediente (ad esempio "crema"), ed il numero di paste che vogliono, e poi il barista cercherà di soddisfare l'ordine. A tal scopo il nostro barista memorizza in un secondo file di testo gli ordini per l'asporto, un ordine su ogni riga, così composto: **l'ingrediente** desiderato (una stringa di al più 127 caratteri utili, senza spazi); a seguire separato da uno spazio il **numero** di paste (un intero). Ad esempio, un tipico ordine sarà per "crema 5", intendendo l'interesse per cinque paste che contengano la crema.
Ad esempio, i file potrebbero avere il seguente contenuto:

"paste.txt":
```
1.60 crema con pinoli; 75
1.60 crema con pistacchio; 85
1.10 salata vuota; 60
2.50 ricciola farcita col prosciutto; 150
1.60 crema con uvetta; 75
1.20 crema; 60
```

"ordini.txt":
```
crema 3
prosciutto 2
pinoli 5
```

Esercizio 1 – Strutture dati Pasta e Ordine, e funzioni di lett./scritt. (mod. element.h e bar.h/.c)

Si definisca una struttura dati **`Pasta`** per memorizzare i dati relativi ad una pasta come sopra descritta (costo, tipo, e calorie). Si definisca poi una struttura dati **`Ordine`** per memorizzare i dati relativi ad un ordine come sopra descritto (ingrediente, numero paste).
Si definisca la funzione:

```
list leggiPaste(char * fileName);
```

che, ricevuto in ingresso il nome di un file che contiene l'elenco delle paste disponibili nel bar, legga da tale file i dati di tutte le paste e li restituisca in una lista di strutture dati di tipo **`Pasta`**. Qualora la funzione incontri dei problemi nella lettura, o vi siano errori nell'apertura del file, si dovrà stampare un messaggio di errore a video, e restituire una lista vuota.
Si definisca poi la funzione:

```
Ordine * leggiOrdini(char * fileName, int * dim);
```

che, ricevuto in ingresso il nome di un file contenente gli ordini, e un intero passato per riferimento, legga i dati contenuti nel file e li restituisca tramite un array di strutture dati di tipo **`Ordine`**, allocato dinamicamente e della dimensione minima necessaria (non è noto a priori quanti ordini siano memorizzati nel file). Tramite l'intero **`dim`** passato per riferimento la funzione dovrà restituire la dimensione dell'array. Qualora vi siano problemi nella lettura, la funzione restituisca un puntatore a NULL e dimensione pari a zero.

Il candidato abbia cura di realizzare nel main opportuni test al fine di verificare il corretto funzionamento delle funzioni di cui sopra, avendo cura di deallocare la memoria, se necessario.

Fondamenti di Informatica T-1, 2019/2020 – Modulo 2

Prova d'Esame 4A di Giovedì 11 Giugno 2020 – tempo a disposizione 2h

Esercizio 2 – Ordinamento e soddisfacibilità di un ordine (moduli element.h/.c e bar.h/.c)

Si definisca la procedura:

```
void ordina(Ordine * v, int dim);
```

che, ricevuta in ingresso un array di strutture dati di tipo **`Ordine`**, e di dimensione **`dim`**, provveda ad ordinare l'array in base al seguente criterio: in ordine decrescente in base al numero di paste richieste; a parità di numero di paste, in ordine crescente lessicografico in base all'ingrediente. Per l'ordinamento si utilizzi un algoritmo a scelta tra quelli visti a lezione.

Si definisca la funzione:

```
int soddisfacibile(list paste, Ordine unOrdine);
```

che, ricevuta in ingresso una lista di strutture dati di tipo **`Pasta`**, e una struttura dati di tipo **`Ordine`**, restituisca un valore interpretabile come vero se l'ordine è soddisfacibile. Un ordine è soddisfacibile se la lista contiene un numero sufficiente di paste il cui tipo comprende l'ingrediente specificato nell'ordine. Ad esempio, se un ordine richiede tre paste con ingrediente "crema", allora andranno bene paste come "crema con uvetta", "pinoli e crema", "panna, crema e chantilly". Si ricorda agli studenti l'esistenza della funzione di libreria **`char * strstr(char * str, char * substr)`** che restituisce un puntatore alla prima occorrenza di **`substr`** in **`str`**, se presente, o **`NULL`** altrimenti.

Esercizio 3 – Registrazione offerta (moduli element.h/.c e palazzo.h/palazzo.c)

Si definisca la funzione:

```
list eseguiOrdine(list paste, Ordine unOrdine);
```

che, ricevuta in ingresso la lista contenente le paste disponibili, e un ordine, verifichi se l'ordine specificato può essere soddisfatto (tramite la funzione di cui al punto precedente).
In caso positivo, la funzione deve restituire una nuova lista contenete le paste disponibili a banco, e ottenuta partendo dalla lista data in ingresso, da cui però siano state rimosse le paste necessarie per soddisfare l'ordine.
In caso negativo, la funzione deve restituire la lista originale (infatti nessuna pasta è stata presa per soddisfare l'ordine).
In altre parole, la funzione simula il fatto che il barista soddisfi/non soddisfi l'ordine, e quindi riporta le paste che rimangono a banco per i successivi clienti e/o ordini.

Esercizio 4 -- Stampa dei risultati, e de-allocazione memoria (main.c)

Il candidato realizzi nella funzione **`main(…)`** un programma che legga dai file i dati relativi alle paste disponibili e agli ordini, provveda ad ordinare l'array degli ordini come specificato all'Esercizio 2, e poi provi a soddisfare gli ordini. Per ogni ordine, il programma stampi un messaggio a video indicando sia se è stato possibile soddisfarlo, sia se non è stato possibile. Infine, il programma stampi a video quante paste sono rimaste a banco al termine.
Al termine del programma, il candidato abbia cura di de-allocare tutta la memoria allocata dinamicamente, ivi compresa la memoria allocata per le liste, se possibile.

Fondamenti di Informatica T-1, 2019/2020 – Modulo 2

Prova d'Esame 4A di Giovedì 11 Giugno 2020 – tempo a disposizione 2h

"element.h":

```
#define _CRT_SECURE_NO_WARNINGS

#ifndef _ELEMENT_H
#define _ELEMENT_H

#include <string.h>

#define DIM_TIPO 256
#define DIM_INGREDIENTE 128

typedef struct {
	float costo;
	char tipo[DIM_TIPO];
	int calorie;
} Pasta;

typedef struct {
	char ingrediente[DIM_INGREDIENTE];
	int numPaste;
} Ordine;

typedef Pasta element;

int compare(Ordine o1, Ordine o2);

#endif
```

"element.c":

```
#include "element.h"

int compare(Ordine o1, Ordine o2) {
	int result;

	result = o2.numPaste - o1.numPaste;
	if (result == 0)
		result = strcmp(o1.ingrediente, o2.ingrediente);
	return result;
}
```

Fondamenti di Informatica T-1, 2019/2020 – Modulo 2

Prova d'Esame 4A di Giovedì 11 Giugno 2020 – tempo a disposizione 2h

"list.h":

```
#pragma once
#ifndef LIST_H
#define LIST_H

#include "element.h"

typedef struct list_element {
	element value;
	struct list_element  *next;
} item;

typedef  item* list;

typedef  int boolean;

/* PRIMITIVE  */
list emptylist(void);
boolean empty(list);
list cons(element, list);
element head(list);
list tail(list);

void showlist(list l);
void freelist(list l);
#endif
```

"list.c:"

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "list.h"

/* OPERAZIONI PRIMITIVE */
list  emptylist(void) {          /* costruttore lista vuota */
	return NULL;
}

boolean  empty(list l) {   /* verifica se lista vuota */
	return (l == NULL);
}

list  cons(element e, list l) {
	list t;       /* costruttore che aggiunge in testa alla lista */
	t = (list)malloc(sizeof(item));
	t->value = e;
	t->next = l;
	return(t);
}

element head(list l) { /* selettore testa lista */
	if (empty(l)) exit(-2);
	else return (l->value);
}

list  tail(list l) {        /* selettore coda lista */
	if (empty(l)) exit(-1);
	else return (l->next);
}

void showlist(list l) {
	element temp;
	if (!empty(l)) {
		temp = head(l);
```

```
                printf("%f %s %d\n",
                        temp.costo, temp.tipo, temp.calorie);
                showlist(tail(l));
                return;
        }
        else {
                printf("\n\n");
                return;
        }
}

void freelist(list l) {
        if (empty(l))
                return;
        else {
                freelist(tail(l));
                free(l);
        }
        return;
}
```

Fondamenti di Informatica T-1, 2019/2020 – Modulo 2

Prova d'Esame 4A di Giovedì 11 Giugno 2020 – tempo a disposizione 2h

"bar.h":

```
#define _CRT_SECURE_NO_WARNINGS

#ifndef _BAR_H
#define _BAR_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "element.h"
#include "list.h"

// Es. 1
list leggiPaste(char * fileName);
Ordine * leggiOrdini(char * fileName, int * dim);

// Es. 2
void ordina(Ordine * v, int dim);
int soddisfacibile(list paste, Ordine unOrdine);

// Es. 3
list eseguiOrdine(list paste, Ordine unOrdine);

#endif
```

"bar.c":

```
#include "bar.h"

// Es. 1
list leggiPaste(char * fileName) {
	list result;
	FILE * fp;
	Pasta temp;
	int letto;
	int i;
	char aChar;

	result = emptylist();

	fp = fopen(fileName, "rt");
	if (fp != NULL) {
		letto = fscanf(fp, "%f", &(temp.costo));
		while (letto == 1) {
			fgetc(fp); // si mangia lo spazio prima della stringa
			i = 0;
			while ( (i<DIM_TIPO-1) && (aChar = fgetc(fp)) != ';') {
				temp.tipo[i] = aChar;
				i++;
			}
			temp.tipo[i] = '\0';
			fscanf(fp, "%d", &(temp.calorie));
			result = cons(temp, result);
			letto = fscanf(fp, "%f", &(temp.costo));
		}
		fclose(fp);
		return result;
	}
	else {
		printf("Errore apertura file %s\n", fileName);
```

```
		return result;
	}
}

Ordine * leggiOrdini(char * fileName, int * dim) {
	Ordine * result;
	FILE * fp;
	int count;
	int letto;
	Ordine temp;

	*dim = 0;
	result = NULL;
	fp = fopen(fileName, "rt");
	if (fp == NULL) {
		printf("Errore paertura file %s\n", fileName);
		return result;
	}
	else {
		// conta degli elementi
		count = 0;
		do {
			letto = fscanf(fp, "%s%d", temp.ingrediente, &(temp.numPaste));
			if (letto == 2)
				count++;
		} while (letto == 2);

		if (count > 0) { // allocazione e poi lettura
			result = (Ordine *)malloc(sizeof(Ordine) * count);
			rewind(fp);
			do {
				letto = fscanf(fp, "%s%d", temp.ingrediente, &(temp.numPaste));
				if (letto == 2) {
					result[*dim] = temp;
					*dim = *dim + 1;
				}
			} while (letto == 2);
		}

		fclose(fp);
		return result;
	}
}

// Es. 2
void scambia(Ordine * e1, Ordine * e2) {
	Ordine tmp = *e1;
	*e1 = *e2;
	*e2 = tmp;
}
void bubbleSort(Ordine v[], int n) {
	int i, ordinato = 0;
	while (n > 1 && !ordinato) {
		ordinato = 1;
		for (i = 0; i < n - 1; i++)
			if (compare(v[i], v[i + 1]) > 0) {
				scambia(&v[i], &v[i + 1]);
				ordinato = 0;
			}
		n--;
	}
}

void ordina(Ordine * v, int dim) {
	bubbleSort(v, dim);
```

```
}

int soddisfacibile(list paste, Ordine unOrdine) {
	int count;
	Pasta temp;

	count = 0;
	while (!empty(paste)) {
		temp = head(paste);
		if (strstr(temp.tipo, unOrdine.ingrediente) != NULL)
			count++;
		paste = tail(paste);
	}
	if (count >= unOrdine.numPaste)
		return 1;
	else
		return 0;
}


// Es. 3
list eseguiOrdine(list paste, Ordine unOrdine) {
	list result;
	int count;
	Pasta temp;

	result = emptylist();

	if (soddisfacibile(paste, unOrdine)) {
		count = unOrdine.numPaste;
		while (!empty(paste) && count>0) {
			temp = head(paste);
			if (strstr(temp.tipo, unOrdine.ingrediente) == NULL)
				result = cons(temp, result);
			else
				count--;
			paste = tail(paste);
		}
	}
	while (!empty(paste)) {
		result = cons(head(paste), result);
		paste = tail(paste);
	}
	return result;
}
```

Fondamenti di Informatica T-1, 2019/2020 – Modulo 2

Prova d'Esame 4A di Giovedì 11 Giugno 2020 – tempo a disposizione 2h

"main.c":

```
#include <stdio.h>
#include <stdlib.h>
#include "element.h"
#include "list.h"
#include "bar.h"

int main() {
	{ // Es. 1
		list banco;
		Ordine * gliOrdini;
		int dim;
		int i;

		banco = leggiPaste("paste.txt");
		showlist(banco);
		gliOrdini = leggiOrdini("ordini.txt", &dim);
		for (i = 0; i < dim; i++) {
			printf("Ordine: %s %d\n", gliOrdini[i].ingrediente,
gliOrdini[i].numPaste);
		}
		freelist(banco);
		free(gliOrdini);
	}
	{ // Es. 2
		list banco;
		Ordine * gliOrdini;
		int dim;
		int i;
		gliOrdini = leggiOrdini("ordini.txt", &dim);
		ordina(gliOrdini, dim);
		for (i = 0; i < dim; i++) {
			printf("Ordine: %s %d\n", gliOrdini[i].ingrediente,
gliOrdini[i].numPaste);
		}
		banco = leggiPaste("paste.txt");
		if (soddisfacibile(banco, gliOrdini[0]))
			printf("L'ordine \"%s %d\" e' soddisfacibile\n",
gliOrdini[0].ingrediente, gliOrdini[0].numPaste);
		else
			printf("L'ordine \"%s %d\" NON e' soddisfacibile\n",
gliOrdini[0].ingrediente, gliOrdini[0].numPaste);
		freelist(banco);
		free(gliOrdini);
	}
	{ // Es. 3 && 4
		list banco;
		list bancoAggiornato;
		list temp;
		Ordine * gliOrdini;
		int dim;
		int i;
		gliOrdini = leggiOrdini("ordini.txt", &dim);
		ordina(gliOrdini, dim);
		banco = leggiPaste("paste.txt");
		bancoAggiornato = banco;
		printf("\n\n");
		for (i = 0; i < dim; i++) {
			if (soddisfacibile(bancoAggiornato, gliOrdini[i])) {
				printf("L'ordine (%s %d), e' soddisfacibile\n",
gliOrdini[i].ingrediente, gliOrdini[i].numPaste);
				temp = bancoAggiornato;
				bancoAggiornato = eseguiOrdine(bancoAggiornato, gliOrdini[i]);
				if (temp != banco)
```

```
                                freelist(temp);
                        }
                        else
                                printf("L'ordine (%s %d), NON e' soddisfacibile\n",
gliOrdini[i].ingrediente, gliOrdini[i].numPaste);
                }
                printf("\n\nDopo aver soddisfatto alcuni ordini, il banco ha le seguenti
paste:\n");
                showlist(bancoAggiornato);
                freelist(banco);
                freelist(bancoAggiornato);
                free(gliOrdini);
        }
        return 0;
}
```

Fondamenti di Informatica T-1, 2019/2020 – Modulo 2

Prova d'Esame 4A di Giovedì 11 Giugno 2020 – tempo a disposizione 2h

"paste.txt":

```
1.60 crema con pinoli; 75
1.60 crema con pistacchio; 85
1.10 salata vuota; 60
2.50 ricciola farcita col prosciutto; 150
1.60 crema con uvetta; 75
1.20 crema; 60
```

"ordini.txt":

```
crema 3
prosciutto 2
pinoli 5
```